

# Graph Algorithms-IV

## **SINGLE-SOURCE SHORTEST PATHS**

---

# Introduction

Generalization of BFS to handle weighted graphs

- ▶ Direct Graph  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ , edge weight fn ;  $w : \mathbf{E} \rightarrow \mathbf{R}$
- ▶ In BFS  $w(e)=1$  for all  $e \in E$

Weight of path  $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$  is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

---

# Shortest Path

Shortest Path = Path of minimum weight

$$\delta(u,v) = \begin{cases} \min \{ \omega(p) : u \stackrel{p}{\rightsquigarrow} v \}; & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

# Shortest-Path Variants

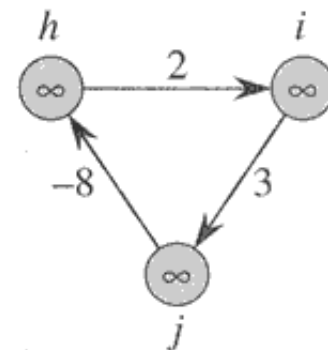
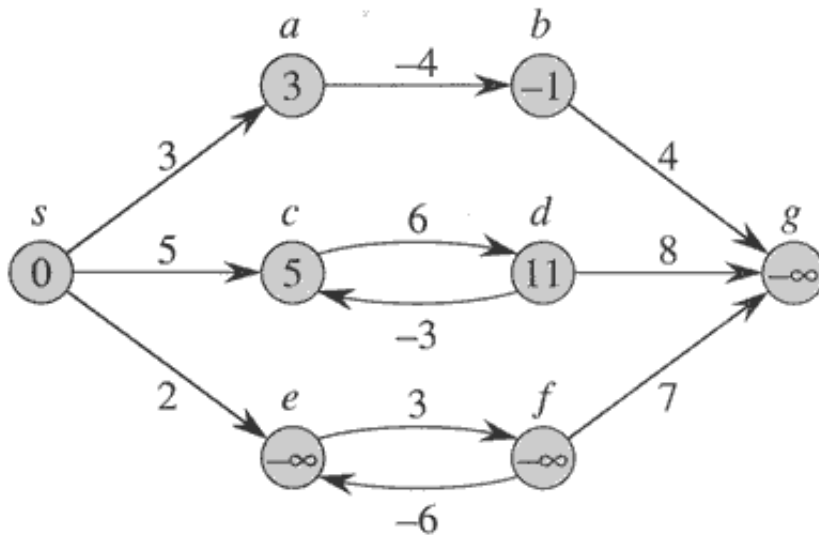
---

## ▶ Shortest-Path problems

- ▶ Single-source shortest-paths problem: Find the shortest path from  $s$  to each vertex  $v$ . (e.g. BFS)
- ▶ Single-destination shortest-paths problem: Find a shortest path to a given *destination* vertex  $t$  from each vertex  $v$ .
- ▶ Single-pair shortest-path problem: Find a shortest path from  $u$  to  $v$  for given vertices  $u$  and  $v$ .
- ▶ All-pairs shortest-paths problem: Find a shortest path from  $u$  to  $v$  for every pair of vertices  $u$  and  $v$ .

# Negative-weight edges

- ▶ No problem, as long as no negative-weight cycles are reachable from the source
- ▶ Otherwise, we can just keep going around it, and get  $w(s, v) = -\infty$  for all  $v$  on the cycle.



# Relaxation

---

- ▶ Maintain  $d[v]$  for each  $v \in V$
- ▶  $d[v]$  is called *shortest-path weight estimate*

*INIT*( $G, s$ )

for each  $v \in V$  do

$d[v] \leftarrow \infty$

$\pi[v] \leftarrow \text{NIL}$

$d[s] \leftarrow 0$

# Relaxation

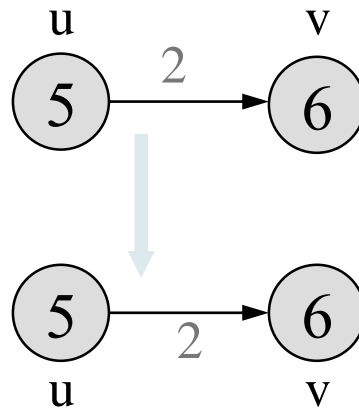
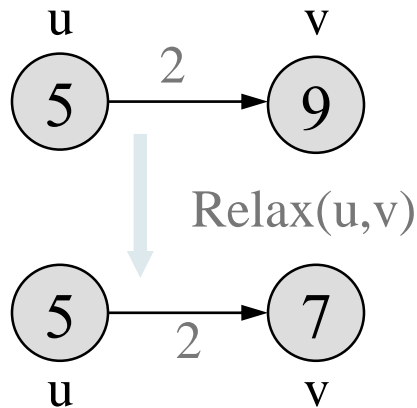
**RELAX**( $u, v$ )

if  $d[v] > d[u] + w(u, v)$  then

$d[v] \leftarrow d[u] + w(u, v)$

$\pi[v] \leftarrow u$

*The process of relaxing an edge( $u, v$ ) consists of testing whether we can improve the shortest path to  $v$  found so far by going through  $u$ . If so, update  $v.d$  and  $v.\pi$*



# Dijkstra's Algorithm For Shortest Paths

---

- ▶ Non-negative edge weight
- ▶ Like BFS: If all edge weights are equal, then use BFS, otherwise use this algorithm
- ▶ Use  $Q$  = priority queue keyed on  $d[v]$  values  
(note: BFS uses FIFO)



# Dijkstra's Algorithm For Shortest Paths

Total time:  $O(V + V \lg V + E \lg V) = O(E \lg V)$

**DIJKSTRA**(G, s)

*INIT*(G, s)

$O(V)$

$S \leftarrow \emptyset$

$Q \leftarrow V[G]$

$O(V)$  if Q is implemented as a min-heap

**while**  $Q \neq \emptyset$  **do**

$u \leftarrow \text{EXTRACT-MIN}(Q)$  Takes  $O(\lg V)$

$S \leftarrow S \cup \{u\}$

**for** each  $v \in \text{Adj}[u]$  **do**

*RELAX*(u, v, w)

Executed  $O(E)$  times

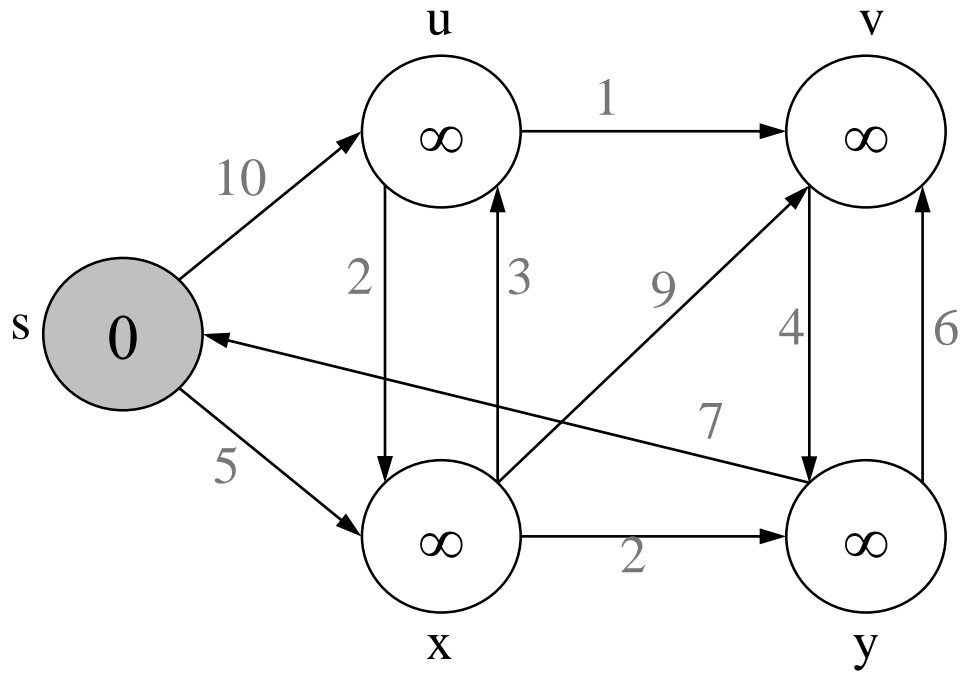
Takes  $O(\lg V)$

Min-heap operations:  
 $O(V \lg V)$

$O(E \lg V)$

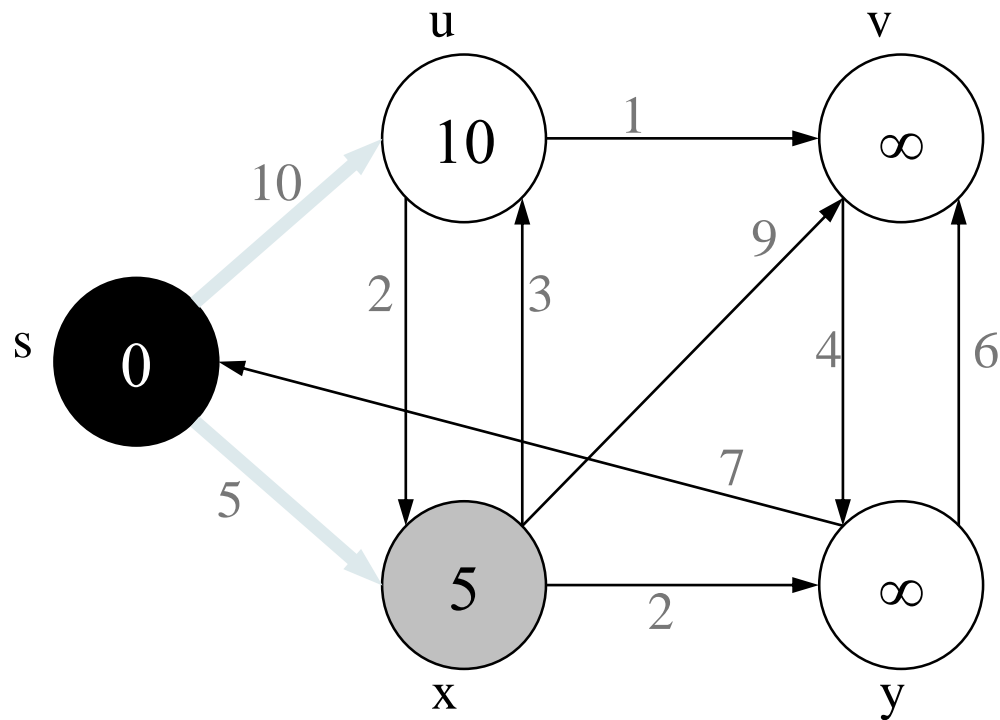
# Example

---



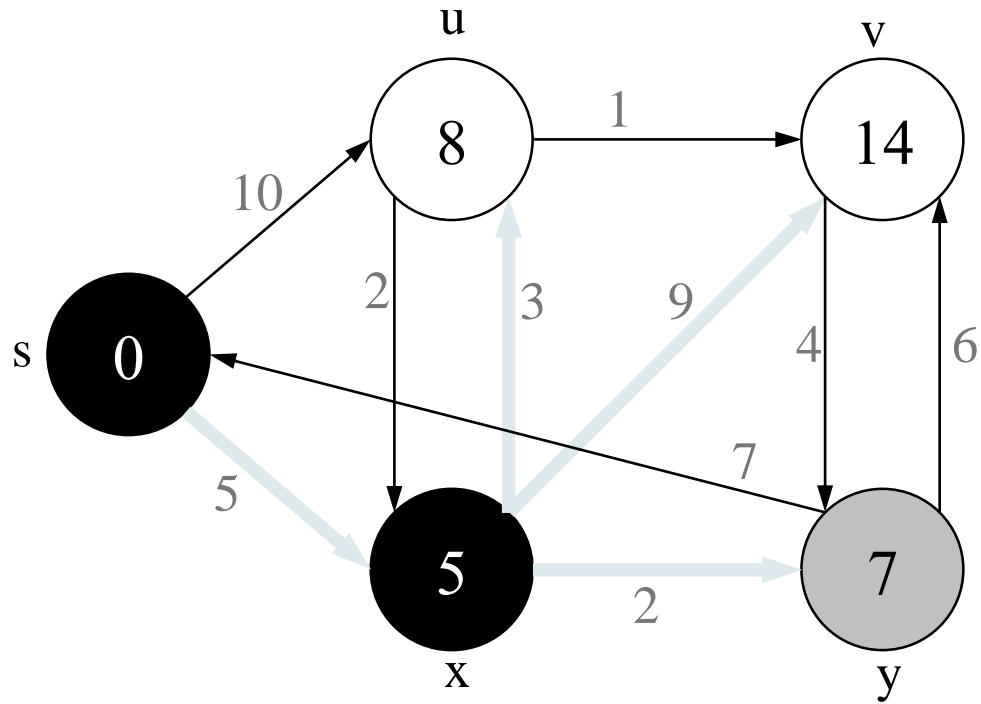
# Example

---



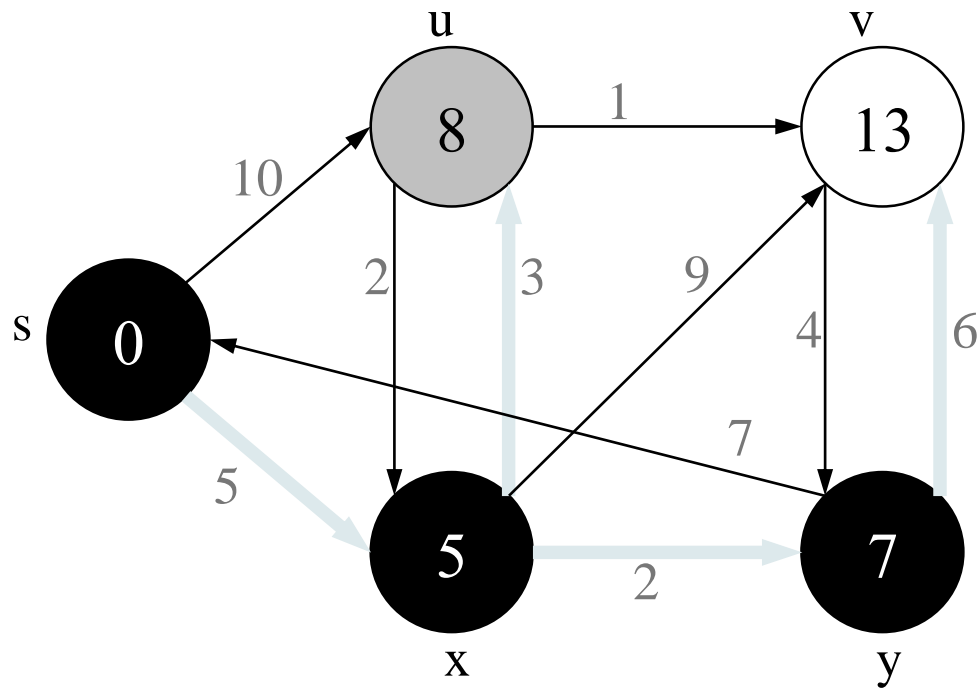
# Example

---



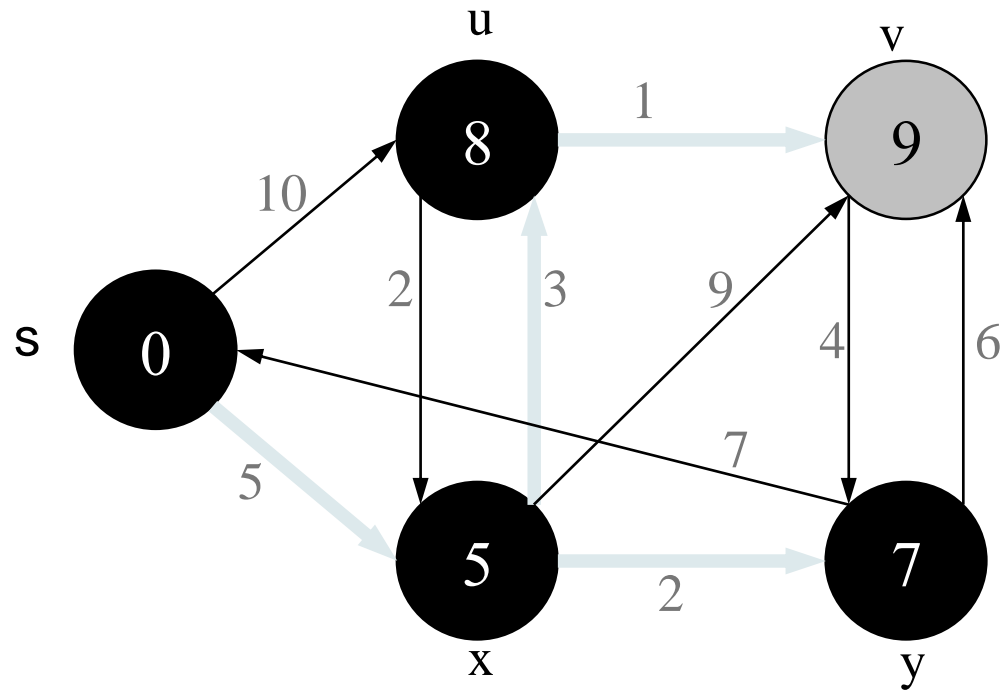
# Example

---



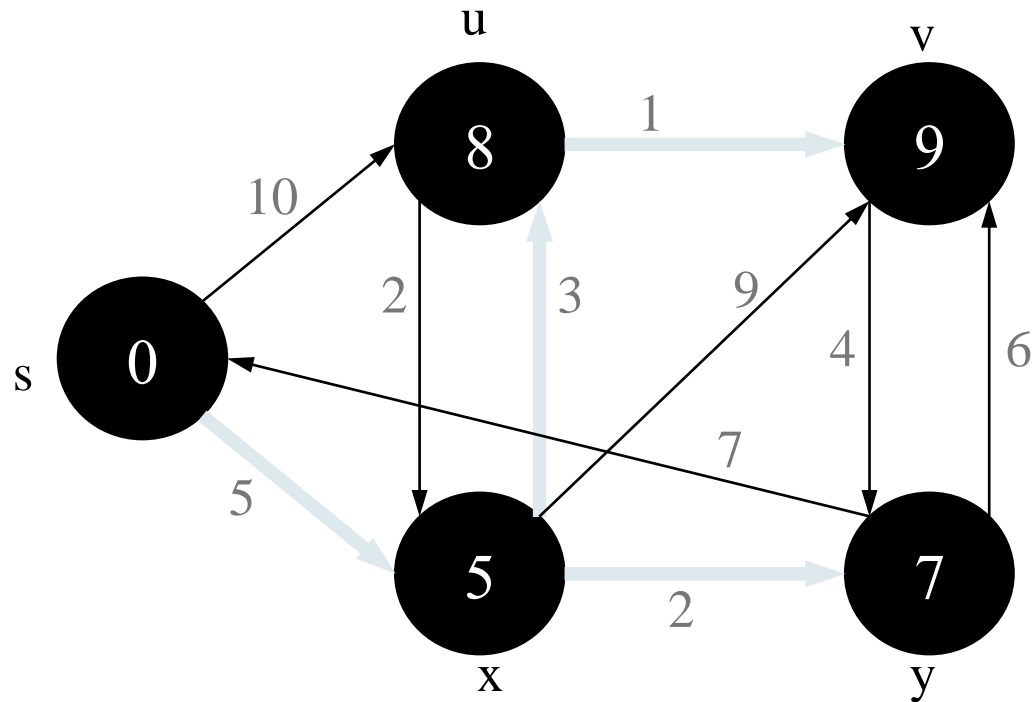
# Example

---



# Example

---



# Dijkstra's Algorithm For Shortest Paths

---

## Observe :

- ▶ Each vertex is extracted from  $Q$  and inserted into  $S$  exactly once
- ▶ Each edge is relaxed exactly once
- ▶  $S$  = set of vertices whose final shortest paths have already been determined
  - i.e.,  $S = \{v \in V: d[v] = \delta(s, v) \neq \infty\}$



# Dijkstra's Algorithm For Shortest Paths

---

- ▶ *Similar to BFS algorithm:* S corresponds to the set of black vertices in **BFS** which have their correct breadth-first distances already computed
- ▶ *Greedy strategy:* Always chooses the **closest(lightest)** vertex in  $Q = V - S$  to insert into S
- ▶ **Relaxation** may reset  $d[v]$  values

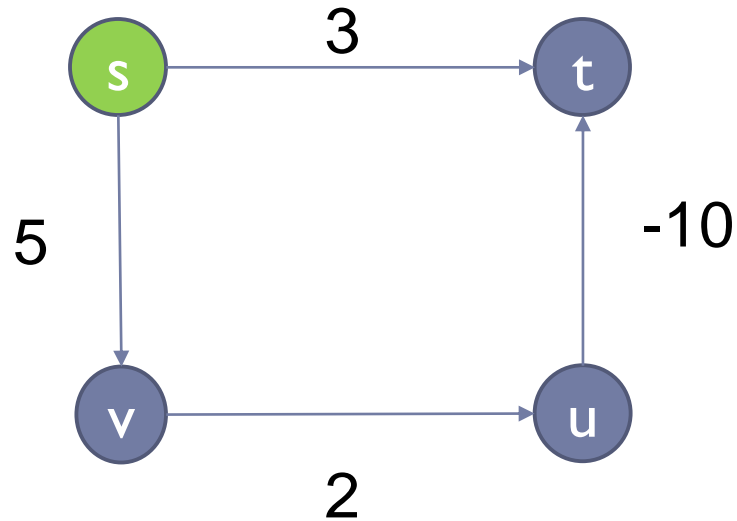
# Dijkstra's Algorithm For Shortest Paths

---

- ▶ **Similar to Prim's MST algorithm:** Both algorithms use a priority queue to find the lightest vertex outside a given set  $S$
- ▶ Insert this vertex into the set
- ▶ Adjust weights of remaining adjacent vertices outside the set accordingly

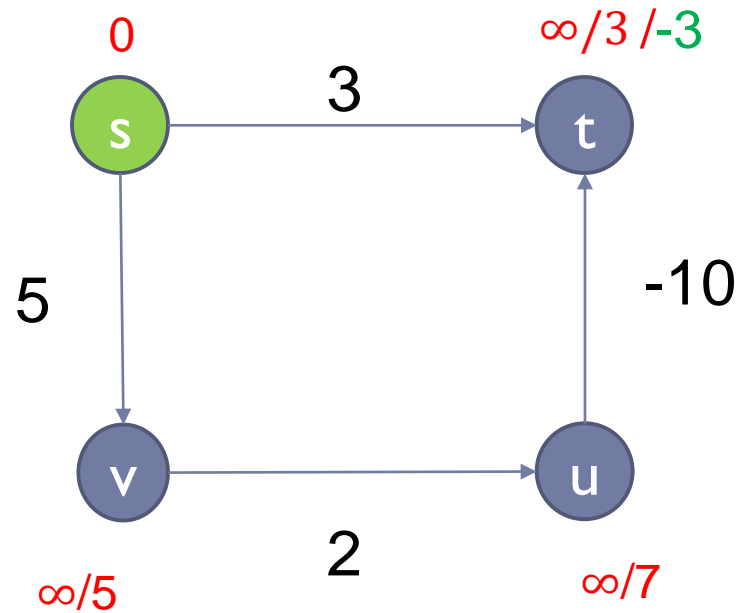
# Dijkstra's Algorithm-Disadvantages

---



# Dijkstra's Algorithm-Disadvantages

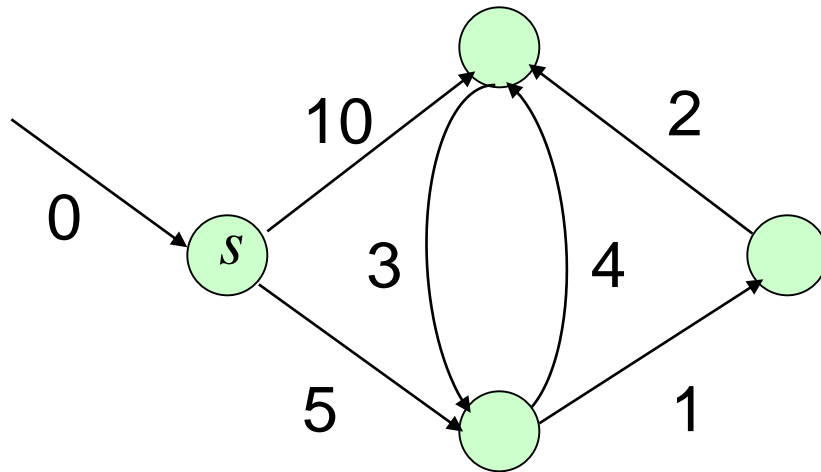
---



# Dijkstra's Algorithm For Shortest Paths

---

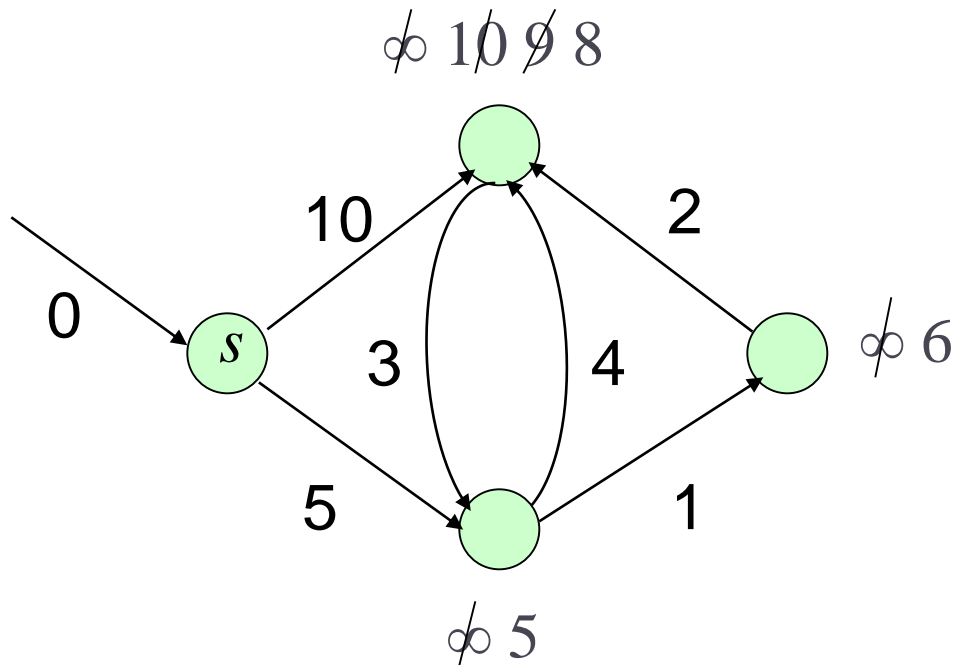
*Example:* Run algorithm on a sample graph



# Dijkstra's Algorithm For Shortest Paths

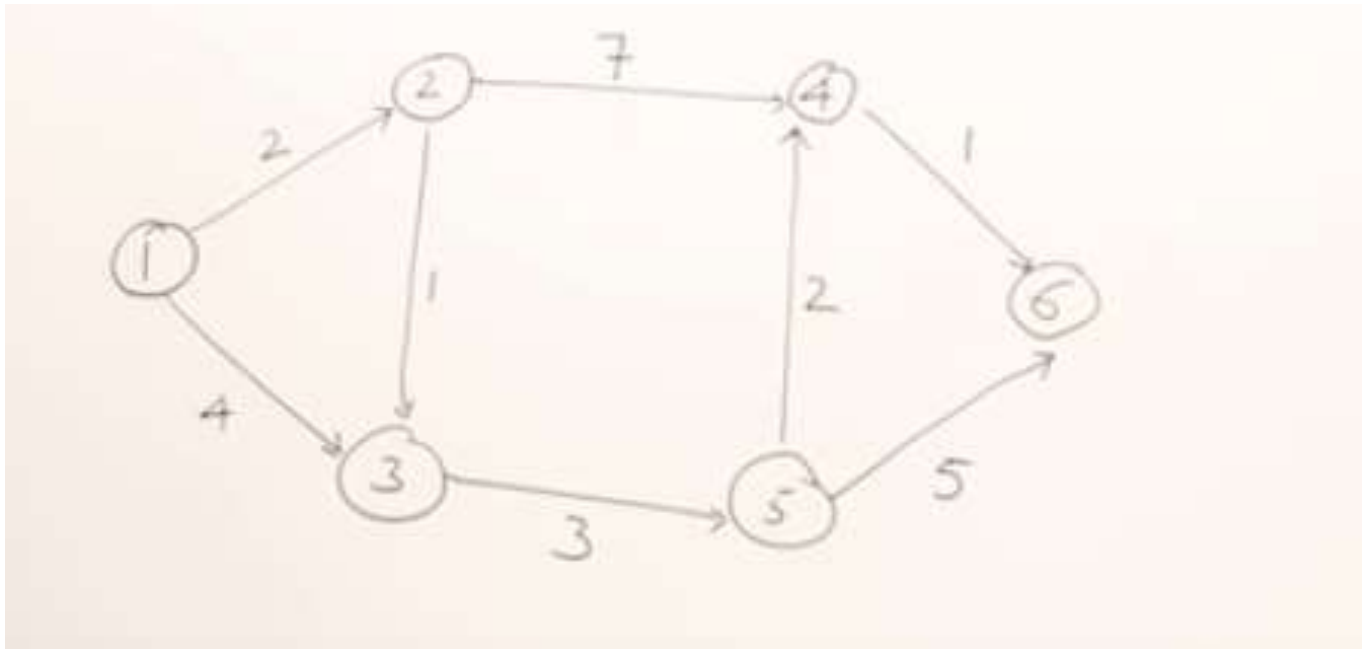
---

**Example:** Run algorithm on a sample graph



# Problem

---



# All-Pair Shortest Path

The Floyd-Warshall Algorithm



# A recursive solution to the all-pairs shortest paths problem:

---

- ▶ Let  $d_{ij}^{(k)}$  be the weight of a shortest path from vertex  $i$  to vertex  $j$  with all intermediate vertices in the set  $\{1, 2, \dots, k\}$ . A recursive definition is given by
- ▶ 
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k=0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$
- ▶ The matrix  $D^{(n)} = (d_{ij}^{(n)})$  gives the final answer- $d_{ij}^{(n)} = \delta(i, j)$  for all  $i, j \in V$ -because all intermediate vertices are in the set  $\{1, 2, \dots, n\}$ .

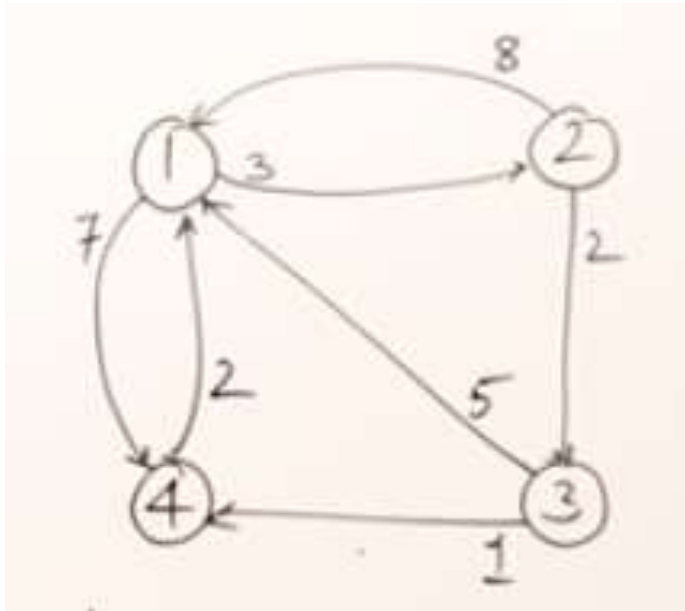
# Computing the shortest-path weights bottom up:

---

- ▶ FLOYD-WARSHALL( $W$ )
- ▶  $n \leftarrow \text{rows}[W]$
- ▶  $D^{(0)} \leftarrow W$
- ▶ **for**  $k \leftarrow 1$  **to**  $n$
- ▶     **do**
- ▶         **for**  $i \leftarrow 1$  **to**  $n$
- ▶             **do for**  $j \leftarrow 1$  **to**  $n$
- ▶                  $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
- ▶ **return**  $D^{(n)}$

# Example:

---



$$A^0 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{matrix}$$

$$A^1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix} \end{matrix}$$

$$A^2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{matrix}$$

$$A^3 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{matrix}$$

$$A^4 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{matrix}$$